A Fast Implementation of the ISODATA Clustering Algorithm*

Nargess Memarsadeghi† David M. Mount‡ Nathan S. Netanyahu§
Jacqueline Le Moigne¶

April 18, 2005

Abstract

Clustering is central to many image processing and remote sensing applications. ISODATA is one of the most popular and widely used clustering methods in geoscience applications, but it can run slowly, particularly with large data sets. We present a more efficient approach to ISODATA clustering, which achieves better running times by storing the points in a kd-tree and through a modification of the way in which the algorithm estimates the dispersion of each cluster. We also present an approximate version of the algorithm which allows the user to further improve the running time, at the expense of lower fidelity in computing the nearest cluster center to each point. We provide both empirical and theoretical justification that our modified approach produces clusterings that are very similar to those produced by the standard ISODATA approach. We also provide empirical studies on both synthetic images and remotely sensed Landsat and Modis images that show that our approach has significantly lower running times.

1 Introduction

Unsupervised clustering is a fundamental tool in image processing for geoscience and remote sensing applications. For example, unsupervised clustering is often used to obtain vegetation maps of an area of interest. This approach is useful when reliable training data are either scarce or expensive, and when relatively little *a priori* information about the data is available. Unsupervised clustering methods play a significant role in the pursuit of unsupervised classification [30].

The problem of clustering points in multidimensional space can be posed formally as one of a number of well known optimization problems, such as the Euclidean *k-median problem* [15], in which the objective is to minimize the sum of distances to the nearest center, the Euclidean *k-center*

*A preliminary version of this paper appeared in the *Proc. IEEE International Geoscience and Remote Sensing Symposium.* (IGARSS'03), Toulouse, France, 2003, Vol. III, 2057–2059.

†NASA Goddard Space Flight Center, Architecture and Automation Branch, Greenbelt, MD 20771 and Department of Computer Science, University of Maryland, College Park, Maryland, 20742. Email: nargess@cs.umd.edu.

‡Department of Computer Science, University of Maryland, College Park, Maryland, 20742. The work of this author was supported by the Science Foundation under grant CCR-0098151. Email: mount@cs.umd.edu.

§Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel and Center for Automation Research, University of Maryland, College Park, Maryland, 20742. Email: nathan@macs.biu.ac.il.

¶NASA Goddard Space Flight Center, Applied Information Sciences Branch, Greenbelt, MD 20771. Email: Jacqueline.J.LeMoigne-Stewart@nasa.gov.

problem [13], objective is to minimize the maximum distance, and the *k-means problem*, in which the objective is to minimize the sum of squared distances [12, 16, 23, 24]. Efficient solutions are known to exist only in special cases, for example, efficient solutions exist for the planar 2-center problem [1, 32]. There are no efficient exact solutions known to any of these problems for general k, and some formulations are known to be NP-hard [11]. Efficient approximation algorithms have been developed in some cases. These include constant factor approximations for the k-center problem [7, 13], the k-median problem [3], and the k-means problem [18]. There are also ϵ-approximation algorithms for k-median [2, 22] and k-means [26], and improvements based on core sets [14]. In spite of their theoretical efficiency, these ϵ-approximation algorithms are not suitable for practical implementation due to the very high constant factors hidden in their asymptotic analyses, except when the dimension k are very small.

In practice, it is common to use heuristic approaches, which seek to find a reasonably good clustering, but do not provide guarantees on the quality of the results. This includes methods such as randomized approaches such as CLARA [20] and CLARANS [27], methods based on neural nets [21]. One of the most popular and widely used clustering heuristics used in remote sensing is ISODATA [4, 16, 17, 33]. A set of n data points in d-dimensional space is given along with an integer k indicating the initial number of clusters and a number of additional parameters. The general goal is to compute a set of cluster centers in d-space. Although there is no specific optimization criterion, the algorithm is similar in spirit to the well known k-means clustering method [16] in which the objective is to minimize the average squared distance of each point to its nearest center, called the *average distortion*. One significant advantages of ISODATA over k-means is that the user need only provide an initial estimate of the number of clusters, and the algorithm may alter the number of clusters by either deleting small clusters, merging nearby clusters, or splitting large diffuse clusters. The algorithm will be described in the next section.

As currently implemented, ISODATA can run very slowly, particularly on large data sets. Given its wide use in remote sensing, its efficient computation is an important goal. Our objective in this paper is not to provide a new or better clustering algorithm, but rather to show how methods from computational geometry can be applied to produce a faster implementation of ISODATA clustering. There are a number of minor variations of ISODATA appearing in the literature. These variations involve issues such as termination conditions, but are equivalent in terms of their overall structure. We focus on a widely used version, called ISOCLUS [28], which will be presented in the next section.

The running times of ISODATA and ISOCLUS are dominated by the time needed to compute the nearest among the k cluster centers to each of the n points. This can be reduced to the problem of answering n nearest neighbor queries over a set of size k, which naively would involve $O(kn)$ time. To improve the running time, an obvious alternative would be to store the k centers in a spatial index, a kd-tree for example [5]. However, this is not the best approach, because k is typically much smaller than n, and the center points are constantly changing, requiring the tree to be constantly updated. Kanungo *et al.* [19] proposed a more efficient and practical approach by storing the points, rather than the cluster centers, in a kd-tree. The tree is then used to solve the *reverse nearest neighbor problem*, that is, for each center we compute the set of points for which this center is the closest. This is called the *filtering algorithm*.

We show how to modify this approach for ISOCLUS. The modifications are not trivial. First off, in order to perform the sort of aggregate processing that the filtering algorithm employs, it was necessary to modify the way in which the ISOCLUS algorithm computes the degree of dispersion

within each cluster. In Sections 5 and 6 we present empirical and theoretical justification that this modification does not significantly alter the nature of the clusters that the algorithm produces. In order to further improve execution times, we have also introduced an approximate version of the filtering algorithm. A user-supplied approximation error bound $\epsilon > 0$ is provided to the algorithm, and each point is associated with a center whose distance from the point is not farther than $(1+\epsilon)$ times the distance to its true nearest neighbor. This result may be of independent interest because it can be applied to k-means clustering as well. It is presented in Section 3.5.

The running time of the filtering algorithm is a subtle function of the structure of the clusters and centers, and so rather than presenting a worst-case asymptotic analysis, we present an empirical analysis of its efficiency based on both synthetically generated data sets, and actual data sets from a common application in geostatistics. These results are presented in Section 5. These experiments show that, depending on the various input parameters (dimension, data size, number of centers), this algorithm runs faster than a straightforward implementation of ISOCLUS, by factors ranging from 1.3 to over 15. In particular, the improvements a very good for typical applications in geostatistics, where the data size n and number of centers k are large, and the dimension d is relatively small. Thus, we feel that this algorithm can be an important tool in the analysis of geostatistical analysis and other applications of data clustering.

2 The ISOCLUS Algorithm

We begin by presenting the particular variant of ISODATA, called ISOCLUS [28], that we will be using. Although our description is not exhaustive, it contains enough information to understand our various modifications. It tries to find the best cluster centers through an iterative approach until some convergence criteria are met. ISOCLUS uses different heuristics to determine when to merge or split clusters.

At a high level, in each iteration of the algorithm the following takes place: points are assigned to their closest cluster centers, cluster centers are updated to be the centroid of their associated points, clusters with very few points are deleted, large clusters satisfying some heuristics are split, and small clusters satisfying other heuristics are merged. The algorithm stops performing iterations until some convergence criteria are met. Here we go over the algorithm in more detail.

There are a number of user-supplied parameters. These include the following. (In parentheses we give the variable name of the parameter used in [28].)

k_{init}: initial number of clusters (NUMCLUS)
$n_{\min}$: minimum number of points that can form a cluster (SAMPRM)
$I_{\max}$: maximum number of iterations (MAXITER)
$\sigma_{\max}$: maximum standard deviation of points from their cluster center along each axis(STDV)
$L_{\min}$: minimum required distance between two cluster centers (LUMP)
$P_{\max}$: maximum number of cluster pairs that can be merged per iteration (MAXPAIR)

Here is an overview of the algorithm. See [28] for details. Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denote the set of points to be clustered. Each point $\mathbf{x}_j = (x_{j1}, \ldots, x_{jd})$ is treated as a vector in real d-dimensional space, $\mathbb{R}^d$. Let n denote the number of points. If the original set is too large, all of the iterations of

the algorithm, except the last, can be performed on a random subset of S of an appropriate size. Throughout, let $\|\mathbf{x}\|$ denote the Euclidean length of the vector $\mathbf{x}$.

(1) Letting $k = k_{\text{init}}$, randomly sample k cluster initial centers $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k\}$ from S.

(2) Assign each point to its closest cluster center. For $1 \le i \le k$, let $S_i \subseteq S$ be the subset of points that are closer to $\mathbf{z}_i$ than to any other cluster center of Z. That is, for any $\mathbf{x} \in S$,

$$\mathbf{x} \in S_j \quad \text{if} \quad \|\mathbf{x} - \mathbf{z}_j\| < \|\mathbf{x} - \mathbf{z}_i\|, \quad \forall i \neq j.$$

(Ties for the closest center are broken arbitrarily.) Let n_j denote the number of points of S_j.

(3) Remove cluster centers with fewer than $n_{\min}$ points. (The associated points of S are not deleted, but are ignored for the remainder of the iteration.) Adjust the value of k and relabel the remaining clusters $S_1 \ldots, S_k$ accordingly.

(4) Move each cluster center to the centroid of the associated set of points. That is,

$$\mathbf{z}_j \leftarrow \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \mathbf{x}, \qquad \text{for } 1 \le j \le k.$$

If any clusters were deleted in Step 3, algorithm goes back to Step 2.

(5) Let Δ_j be the average distance of points of S_j to the associated cluster center $\mathbf{z}_j$ and let Δ be the overall average of these distances.

$$\Delta_j \leftarrow \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|, \quad \text{for } 1 \le j \le k. \qquad\qquad \Delta \leftarrow \frac{1}{n} \sum_{j=1}^{k} n_j \Delta_j.$$

(6) If this is the last iteration, set $L_{\min} = 0$ and go to Step 9. Also, if $2k > k_{\text{init}}$ and it is either an even numbered iteration or $k \ge 2k_{\text{init}}$, go to Step 9.

(7) For each cluster S_j, compute a vector $\mathbf{v}_j = (v_1, \ldots, v_d)$ whose ith coordinate is the standard deviation of the ith coordinates of the vectors directed from $\mathbf{z}_j$ to every point of S_j. That is,

$$v_{ji} \leftarrow \left(\frac{1}{n_j} \sum_{\mathbf{x} \in S_j} (x_i - z_{ji})^2 \right)^{1/2} \qquad \text{for } 1 \le j \le k \text{ and } 1 \le i \le d.$$

Let $v_{j,\max}$ denote the maximum coordinate of $\mathbf{v}_j$.

(8) For each cluster S_j, if $v_{j,\max} > \sigma_{\max}$ and either:

$$((\Delta_j > \Delta) \ \text{ and } \ (n_j > 2(n_{\min} + 1))) \ \text{ or } \ k \le k_{\text{init}}/2$$

then split S_j into two clusters by replacing its center with two cluster centers centered around $\mathbf{z}_j$ and separated by an amount and direction that depends on $v_{j,\max}$ [28]. If any clusters are split in this step, go to Step 2.

(9) Compute the pairwise *intercluster distances* between all distinct pairs of cluster centers:

$$d_{ij} \leftarrow \|\mathbf{z}_i - \mathbf{z}_j\|, \qquad \text{for } 1 \le i < j \le k.$$

(10) Sort the intercluster distances of Step 9 in increasing order, and select a subset of at most $P_{\max}$ of the closest such pairs of clusters such that each pair has an intercluster distance of at most $L_{\min}$. For each such pair (i, j), if neither S_i nor S_j has been involved in a merger in this iteration, replace the two clusters S_i and S_j with a merged cluster $S_i \cup S_j$, whose associated cluster center is their weighted average:

$$\mathbf{z}_{ij} \leftarrow \frac{1}{n_i + n_j}(n_i \mathbf{z}_i + n_j \mathbf{z}_j).$$

Relabel the remaining clusters and reduce k accordingly.

(11) If the number of iterations is less than $I_{\max}$, go to Step 2.

If the algorithm is implemented in the most straightforward manner, and if it is assumed that number of clusters, k, is much smaller than total number of points, n, then the most time consuming part of the algorithm is Step 2, which naively would appear to involve computing the distance from each of the n points of S to each of the k centers, would involve a total of $O(kn)$ time (assuming the dimension d is fixed.) Our approach to improving the algorithm's running time is to speed up this step through the use of an appropriate data structure. Note as well that the algorithm does not even necessarily need to know explicitly how to compute the closest center to each point. The information that the algorithm really needs is the centroid of the points that are closest to each center. Our focus is to compute this quantity efficiently. Before doing this, we provide some background on a related clustering algorithm called Lloyd's algorithm and a fast implementation by a method called the filtering algorithm.

3 The Filtering Algorithm

At its heart, the ISOCLUS algorithm is based on an enhancement of a simple and widely used heuristic for k-means clustering, sometimes called *Lloyd's algorithm* or the *k-means algorithm* [9, 23, 24]. It iteratively repeats the following two steps until convergence. First, for each cluster center, it computes the set of points for which this center is the closest. Next, it moves each center to the centroid of its associated set. It can be shown that with each step the average distortion decreases and that the algorithm converges to a local minimum [31]. See [6, 25, 29] for further discussion of the statistical and convergence properties of Lloyd's and related algorithms. The ISOCLUS algorithm combines Lloyd's algorithm with additional mechanisms for eliminating very small clusters (Step 3), splitting large clusters (Steps 7–8), and merging nearby clusters (Steps 9–10).

As with ISOCLUS, the running time of Lloyd's algorithm is dominated by the time to compute the nearest cluster center to each data point. Naively, this would require $O(kn)$ time. Kanungo *et al.* [19] presented a more efficient implementation of Lloyd's algorithm, called the *filtering algorithm*. Although its worst-case asymptotic running time is not better than the naive algorithm, this approach was shown to be quite efficient in practice. In this section we present a high-level description of this algorithm. We also introduce an approximation version of this algorithm, in which points may be assigned not to their nearest neighbor but to an approximate nearest neighbor. This allows the algorithm to trade off accuracy for efficiency.

3.1 The kd-tree

If considered at a high level, the filtering process implicitly involves computing, for each of the k centers, some aggregate information for all the points that are closer to this center than any other. In particular, it needs to compute the centroid of these points and some other statistical information that is used by the ISOCLUS algorithm. Thus, the process can be viewed very abstractly as answering a number of range queries involving k disjoint ranges, each being the Voronoi cell of some cluster center. As such, an approach based on hierarchical spatial subdivisions is natural.

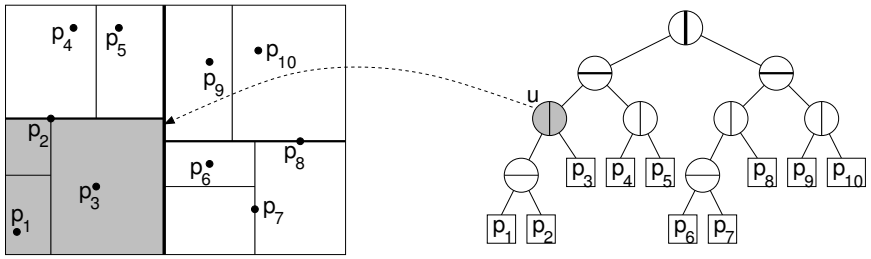

Fig. 1: An example of a kd-tree of a set of points in the plane, showing both the associated spatial subdivision (left) and the binary tree structure (right).

The filtering algorithm builds a standard kd-tree [5], augmented with additional statistical information, which will be discussed below. A *kd-tree* is a hierarchical decomposition of space axis-aligned hyperrectangles called *cells*. Each node of the tree is implicitly associated with a unique cell and the subset of the points that lie within this cell. Each *internal node* of the kd-tree stores an axis-orthogonal *splitting hyperplane*. This hyperplane subdivides the cell into two subcells, which are associated with the left and right subtrees of the node. Nodes holding a single point are declared to be *leaves* of the tree. In Fig. 1, the highlighted node u of the tree is associated with the shaded rectangular cell shown on the left side of the figure and the subset $\{p_1, p_2, p_3\}$ of points. It is well known that a kd-tree on n points can be constructed in $O(n \log n)$ time [10].

3.2 The Filtering Process

We provide an overview how the filtering algorithm is used to perform one iteration of Lloyd's algorithm. (See [19] for details.) Given a kd-tree for the data points S and the current set of k center points. It processes the nodes of the kd-tree in a top-down recursive manner, starting at the root. Consider some node u of the tree. Let $S(u)$ denote the subset of points S that are associated with this node. If it can be inferred that all the points of $S(u)$ are closer to some center $\mathbf{z}_j$ than to any other center (that is, the node's associated rectangular cell lies entirely within the Voronoi cell of $\mathbf{z}_j$) we may *assign* u to cluster S_j. Every point associated with u is thus *implicitly assigned* to this cluster. (This is the case for the node associated with cell a in Fig. 2.) If this cannot be inferred, then the cell is split, and we apply the process recursively on its two children. (This is the case for the node associated with cell b in the figure, which is split and whose two children are b_1 and b_2.) Finally, if the process arrives at a leaf node, which contains a single point, then we determine which center is closest to the point of the associated node, and assign it to this center. (This is the case for the node associated with cell c of the figure.)

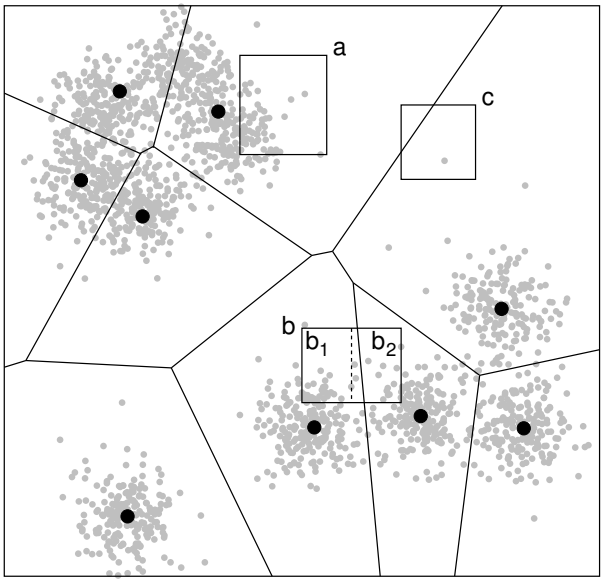

Fig. 2: Classifying nodes in the filtering algorithm. The subdivision is the Voronoi diagram of the centers, which indicates the neighborhood regions of each center.

At the conclusion of the process, the filtering algorithm assigns the nodes of the kd-tree to clusters in such a manner that every point of S is implicitly assigned to its closest cluster center. Furthermore, this is done so that the sets $S(u)$ assigned to a given cluster form a disjoint union of the associated cluster. There are two issues to be considered: (1) how is it determined that one center is closer than all others to every point of a node's cell, and (2) when this occurs, how are the points of the node assigned *en masse* to this center. We address these issues in reverse order in the next two sections.

3.3 Additional Statistical Information

As mentioned above, the k-means algorithm seeks a placement of the centers that minimizes the average squared distance of each point to its nearest center, called the average distortion. For each cluster S_j, recall that $n_j = |S_j|$, and define the *average distortion* of the j^{th} cluster, denoted $\Delta_j^{(2)}$, to be the average squared distance of each point in cluster S_j to its cluster center, that is,

$$\Delta_j^{(2)} = \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|^2.$$

(Contrast this quantity with the average distance Δ_j, computed in Step 5 of the ISOCLUS algorithm.) The overall distortion for the entire data set is the weighted average distortion among all clusters, where the weight factor for the jth cluster is, n_j/n, the fraction of points in this cluster.

In order to provide calculating this information efficiently for each cluster, we store the following statistical information with each node u of the kd-tree. Recall that each point of the data set is represented as a coordinate vector in $\mathbb{R}^d$.

$\mathbf{s}(u)$: or *weighted centroid*, contains the vector sum of the points associated with this node.

$ss(u)$: or *sum of squares*, contains the sum of the dot products $(\mathbf{x} \cdot \mathbf{x})$ for all points $\mathbf{x}$ associated with this node.

$w(u)$: or *weight*, contains the number of points associated with this node.

These quantities can be computed by a simple post-order traversal of the kd-tree in the same $O(dn)$ time. We omit the straightforward details. The following lemma shows that, once we have determined the set of nodes associated with a given center, we can compute the centroid of the set and the distortion of the resulting cluster.

Lemma 3.1 *Consider a fixed cluster S_j, and let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of nodes that are assigned to this cluster, so that S_j is the disjoint union of $S(u_i)$, for $1 \le i \le m$. Consider the sums of these associated node quantities:*

$$\mathbf{s}_j = \sum_{i=1}^{m} \mathbf{s}(u_i) \qquad ss_j = \sum_{i=1}^{m} ss(u_i) \qquad w_j = \sum_{i=1}^{m} w(u_i).$$

Then the size of the cluster is $n_j = w_j$, the centroid of the cluster is $(1/n_j)\mathbf{s}_j$, and the average distortion of the cluster is

$$\Delta_j^{(2)} = \frac{1}{w_j} ss_j - \frac{2}{w_j}(\mathbf{z}_j \cdot \mathbf{s}_j) + (\mathbf{z}_j \cdot \mathbf{z}_j).$$

Proof: Because $\cup_{i=1}^{m} S(u_i)$ is a disjoint partition of S we have

$$\mathbf{s}_j = \sum_{\mathbf{x} \in S_j} \mathbf{x}, \qquad ss_j = \sum_{\mathbf{x} \in S_j} (\mathbf{x} \cdot \mathbf{x}), \qquad w_j = \sum_{\mathbf{x} \in S_j} 1 = |S_j| = n_j.$$

The first two results follow directly, leaving only the average distortion. In a slight abuse of notation, for two vectors $\mathbf{x}$ and $\mathbf{z}$, we express their dot products as $\mathbf{x}^2 = (\mathbf{x} \cdot \mathbf{x})$ and $\mathbf{xz} = (\mathbf{x} \cdot \mathbf{z})$. Then we can express the total distortion for the jth cluster as:

$$\begin{aligned} n_j \Delta_j^{(2)} &= \sum_{\mathbf{x} \in S_j} (\mathbf{x} - \mathbf{z}_j)^2 = \sum_{\mathbf{x} \in S_j} (\mathbf{x}^2 - 2\mathbf{x}\mathbf{z}_j + \mathbf{z}_j^2) = \sum_{\mathbf{x} \in S_j} \mathbf{x}^2 - \sum_{\mathbf{x} \in S_j} 2\mathbf{x}\mathbf{z}_j + \sum_{\mathbf{x} \in S_j} \mathbf{z}_j^2 \\ &= \sum_{\mathbf{x} \in S_j} (\mathbf{x} \cdot \mathbf{x}) - 2 \left(\mathbf{z}_j \cdot \sum_{\mathbf{x} \in S_j} \mathbf{x} \right) + w_j (\mathbf{z}_j \cdot \mathbf{z}_j) \\ &= ss_j - 2(\mathbf{z}_j \cdot \mathbf{s}_j) + w_j (\mathbf{z}_j \cdot \mathbf{z}_j). \end{aligned}$$

The final result follows by dividing by $n_j = w_j$. □

3.4 Assigning Nodes to Centers

All that remains is to explain how the filtering algorithm assigns nodes to each of the cluster centers. Recall that the input to the algorithm is the set S given in the form of a kd-tree, the statistical quantities $\mathbf{s}(u)$, $ss(u)$, and $w(u)$ for each node u of the kd-tree, and the locations of the cluster

centers $\mathbf{z}_j$. As the algorithm assigns a node u to a center $\mathbf{z}_j$, it adds these three quantities to the associated sums $\mathbf{s}_j$, ss_j, and w_j, as defined in the proof of Lemma 3.1. On termination of the algorithm, each center $\mathbf{z}_j$ is associated with the sum of these quantities for all the points S_j.

As mentioned above, the filtering algorithm visits the nodes of the tree in a recursive top-down manner. For each node it visits, it maintains the subset of centers, called *candidates*, such that the closest center to any point in the node's cell is one of these candidate centers. Thus, for each node we keep track of a subset of centers that may serve as the nearest center for any point within the cell. Unfortunately, we know of no sufficiently efficient test to determine the set of true candidates (which involves determining the set of Voronoi cells overlapped by an axis-aligned rectangle). Instead we will describe a simple procedure that associates each node with a superset of its true candidates.

To start the process off, the candidates for the root node of the kd-tree consists of all k centers. The centers are then filtered through the kd-tree as follows. Let C be the cell associated with the current node u, and let Z be the set of candidate centers associated with C. First, the closest center $\mathbf{z}^* \in Z$ to the midpoint of C is computed. Then, for the rest of candidates $\mathbf{z} \in Z \backslash \mathbf{z}^*$, if all parts of C are farther from $\mathbf{z}$ than they are to $\mathbf{z}^*$, we may conclude that $\mathbf{z}$ cannot serve as the nearest center for any point in u. So, we can eliminate, or *filter*, $\mathbf{z}$ from set of candidates. If there is only one candidate center (that is, $|Z| = 1$), then this node is assigned to this center. In particular, this means that the quantities $\mathbf{s}$, ss, and w for node u are added to the corresponding sums for this center. Otherwise, for an internal node, we pass the surviving set of candidates to its two children, and repeat the process recursively. If the algorithm reaches a leaf node having two or more candidates, the distances from all centers of Z to the node's data point are calculated, and this data point is assigned to the nearest candidate center.

In order to determine whether any part of C is closer to candidate $\mathbf{z}$ than to $\mathbf{z}^*$ we proceed as follows. Let H be the hyperplane bisecting the line segment $\overline{\mathbf{z}\mathbf{z}^*}$. (See Fig. 3.) We can filter $\mathbf{z}$ if C is entirely on the same side of H as $\mathbf{z}^*$. This condition is tested through the use of a vector $\mathbf{w} = \mathbf{z} - \mathbf{z}^*$, from $\mathbf{z}^*$ to $\mathbf{z}$. Let $\mathbf{v}$ be the vertex of C that maximizes the dot product $(\mathbf{v} \cdot \mathbf{w})$, that is $\mathbf{v}$ is farthest vertex in C in direction of $\mathbf{w}$. If $\text{dist}(\mathbf{z}, \mathbf{v}) \geq \text{dist}(\mathbf{z}^*, \mathbf{v})$, then $\mathbf{z}$ is pruned. The choice of the vertex $\mathbf{v}$ can be determined simply by signs of the individual coordinates of $\mathbf{w}$. (See [19] for details.) The process requires $O(d)$ time for each center tested.

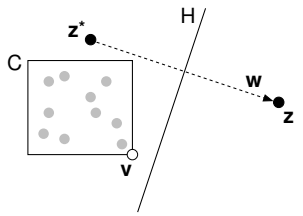

Fig. 3: Filtering process where $\mathbf{z}$ is pruned.

The filtering algorithm achieves its efficiency by assigning many points at once to each center. A straightforward implementation of Lloyd's algorithm requires $O(kn)$ time to compute the distance from each of the n points to each of the k centers. The corresponding measure of complexity for the filtering algorithm is the number of interactions between nodes and candidates. Kanungo *et al.* [19] have shown experimentally that this number is smaller by factors ranging from 10 to 200

for low dimensional clustered data sets. Even considering the additional preprocessing time and overhead, the speed-ups in actual CPU time can be quite significant.

3.5 Approximate Filtering

As with many approaches based on spatial subdivision methods, the filtering algorithm suffers from the curse of dimensionality. This means that, as the dimension increases, the algorithm's running time increases exponentially as a function of dimension. Our approach for dealing with this for data sets of larger dimensionality is to apply filtering in an approximate manner, and so to trade accuracy for speed. In our case, we allow the user to provide a parameter $\epsilon > 0$, and the filtering algorithm is permitted to assign each point of S to any center points that is within distance up to $(1+\epsilon)$ times the distance to the closest center.

This can be incorporated into the filtering process as follows. Recall from the description of the previous section that u is the current node being processed, C denotes u's associated cell, Z denotes u's candidate centers, and $\mathbf{z}^* \in Z$ is the closest center of Z to the midpoint of C. Our goal is to determine those points $\mathbf{z} \in Z \backslash \{\mathbf{z}^*\}$, such that for every point $x \in C$, we have $\|x\mathbf{z}^*\| \leq (1+\epsilon)\|\mathbf{xz}\|$. All such points $\mathbf{z}$ can be filtered. In geometric terms, this is equivalent to replacing the bisector test used in the exact algorithm with a test involving an approximate bisector, denoted $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$, defined to be set of points x such that $\|\mathbf{xz}^*\| = (1+\epsilon)\|\mathbf{xz}\|$. (See Fig. 4.)

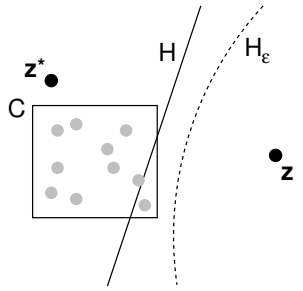

Fig. 4: Approximate filtering, where $\mathbf{z}$ is pruned.

The hyperplane bisector test of the previous section must be adapted to determine whether C is stabbed by $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$. At first, this might seem to be a much harder test to perform. For example, it is no longer sufficient to merely test an appropriate vertex of C, since it is possible that the approximate bisector intersects the interior of a facet of C, while all the vertices lie to one side of the approximate bisector. What saves the day is the fact that the approximate bisector is a hypersphere, and hence the problem reduces to computing the distance between an axis-aligned rectangle, and the center of this hypersphere, which can be computed easily. For completeness, we present the following two technical lemmas, which provide the necessary groundwork.

Lemma 3.2 *Given $\epsilon > 0$, and two points $\mathbf{z}$ and $\mathbf{z}^*$ in d-space, $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$ is a $(d-1)$-sphere of radius r_ϵ centered at the point $\mathbf{c}_\epsilon$, where*

$$r_\epsilon = \frac{1+\epsilon}{\gamma - 1}\|\mathbf{zz}^*\| \qquad \textit{and} \qquad \mathbf{c}_\epsilon = \frac{1}{\gamma-1}(\gamma\mathbf{z} - \mathbf{z}^*), \qquad \textit{where} \qquad \gamma = (1+\epsilon)^2.$$

Proof: A point $\mathbf{x}$ lies on H_ϵ if and only if

$$\|\mathbf{x}\mathbf{z}^*\|^2 = (1+\epsilon)^2\|\mathbf{x}\mathbf{z}\|^2.$$

As before, it will be convenient to express dot products using $\mathbf{x}^2 = (\mathbf{x}\cdot\mathbf{x})$ and $\mathbf{x}\mathbf{z} = (\mathbf{x}\cdot\mathbf{z})$. The above is equivalent to

$$\begin{aligned}(\mathbf{x}-\mathbf{z}^*)^2 &= (1+\epsilon)^2(\mathbf{x}-\mathbf{z})^2\\ \mathbf{x}^2 - 2\mathbf{x}\mathbf{z}^* + \mathbf{z}^{*2} &= \gamma(\mathbf{x}^2 - 2\mathbf{x}\mathbf{z} + \mathbf{z}^2).\end{aligned}$$

Expanding and completing the square yields

$$\begin{aligned}(\gamma-1)\mathbf{x}^2 - 2(\gamma\mathbf{z}-\mathbf{z}^*)\mathbf{x} + (\gamma\mathbf{z}^2 - \mathbf{z}^{*2}) &= 0\\ \mathbf{x}^2 - \frac{2}{\gamma-1}(\gamma\mathbf{z}-\mathbf{z}^*)\mathbf{x} + \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z}-\mathbf{z}^*)^2 &= \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z}-\mathbf{z}^*)^2 - \frac{1}{\gamma-1}(\gamma\mathbf{z}^2-\mathbf{z}^{*2})\\ \left(\mathbf{x} - \frac{1}{\gamma-1}(\gamma\mathbf{z}-\mathbf{z}^*)\right)^2 &= \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z}-\mathbf{z}^*)^2 - \frac{1}{\gamma-1}(\gamma\mathbf{z}^2-\mathbf{z}^{*2}).\end{aligned}$$

The left-hand side is $(\mathbf{x}-c_\epsilon)^2$. Expanding the right-hand side gives

$$\begin{aligned}(\mathbf{x}-c_\epsilon)^2 &= \frac{1}{(\gamma-1)^2}((\gamma\mathbf{z}-\mathbf{z}^*)^2 - (\gamma-1)(\gamma\mathbf{z}^2-\mathbf{z}^{*2}))\\ &= \frac{1}{(\gamma-1)^2}((\gamma^2\mathbf{z}^2 - 2\gamma\mathbf{z}\mathbf{z}^* + \mathbf{z}^{*2}) - (\gamma^2\mathbf{z}^2 - \gamma\mathbf{z}^{*2} - \gamma\mathbf{z}^2 + \mathbf{z}^{*2}))\\ &= \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z}^2 - 2\gamma\mathbf{z}\mathbf{z}^* + \gamma\mathbf{z}^{*2})\\ &= \frac{\gamma}{(\gamma-1)^2}(\mathbf{z}-\mathbf{z}^*)^2 \;=\; \left(\frac{1+\epsilon}{\gamma-1}\|\mathbf{z}\mathbf{z}^*\|\right)^2 \;=\; r_\epsilon^2.\end{aligned}$$

This is the equation of the desired hypersphere. □

Lemma 3.3 *The closest (Euclidean) distance between an axis-aligned hyperrectangle in $\mathbb{R}^d$ and any point $\mathbf{c}\in\mathbb{R}^d$ can be computed in $O(d)$ time.*

Proof: Let $\mathbf{v} = (v_1,\ldots,v_d)$ and $\mathbf{w} = (w_1,\ldots,w_d)$ be the rectangle vertices with the lowest and highest coordinate values, respectively. (For example, these would be the lower left and upper right vertices in the planar case.) The rectangle is just the d-fold intersection of axis-orthogonal strips

$$\{(x_1,\ldots,x_d) \mid v_i \le x_i \le w_i\}.$$

Based on the location of $\mathbf{c}$ relative to each of these strips, we can compute the squared distance from $\mathbf{c} = (c_1,\ldots,c_d)$ to the rectangle as $\sum_{i=1}^d \delta_i^2$, where

$$\delta_i = \begin{cases} v_i - c_i & \text{if } c_i < v_i\\ 0 & \text{if } v_i \le c_i \le w_i\\ c_i - w_i & \text{if } w_i < c_i.\end{cases}$$

The final distance is the square root of this sum. □

Using these two lemmas, it is now easy to see how to replace the exact filtering step described in the previous section with an approximate filtering test, which also runs in $O(d)$ time. Given candidate centers $\mathbf{z}$ and $\mathbf{z}^*$, we apply Lemma 3.2 to compute r_ϵ and $\mathbf{c}_\epsilon$. We then apply Lemma 3.3 to compute the closest distance between the cell C and $\mathbf{c}_\epsilon$. If this distance is greater than r_ϵ then $\mathbf{z}$ is pruned. The remainder of the algorithm is the same. In Section 5.3 below, we present experimental evidence for the benefits of using approximate filtering.

Although points are assigned to cluster centers that are ϵ-nearest neighbors, it does not follow that the result produced by the approximate version of the ISOCLUS algorithm results in an ϵ-approximation in the sense of distortion. The reason is that ISOCLUS is a heuristic and does provide any guarantees on the resulting distortion. It follows some path in the space of possible solutions to some local minimum. Even a minor change to the algorithm's definition can alter this path, and may lead to a local minimum of a significantly different value, either larger or smaller.

4 Our Modifications and Improvements

As mentioned earlier, most of the computational effort in the ISOCLUS algorithm is spent calculating and updating distances and distortions in Steps 2–5. These steps take $O(kn)$ time, whereas all the other steps can be performed in $O(k)$ time, where k is the current number of centers. Our improvement is achieved by adapting the filtering algorithm to compute the desired information. This is the reason for computing the additional statistical information, which was described in the previous section.

There is one wrinkle, however. The filtering algorithm achieves its efficiency by processing points in groups, rather than individually. This works fine as along as the statistical quantities being used by the algorithm can be computed in an aggregated manner. This is true for the centroid, as shown in Lemma 3.1, since it involves the sum of coordinates. Generally, the filtering method can be applied to any polynomial function of the point and center coordinates. However, there is one statistical quantity computed by the ISOCLUS algorithm that does not satisfy this property. In particular, Step 5 of the ISOCLUS algorithm involves computing the sum of Euclidean distances from each point to its closest center as a measure of the dispersion of the cluster. This information is used later in Step 8 to determine whether to split the cluster. This involves computing the sum of square roots, and we know of no way to aggregate this processing.

Rather than implementing ISOCLUS exactly as described in [28], we modified Step 5. For each cluster j, rather than computing the average Euclidean distance of each point to its center, Δ_j, we instead computed the average *squared Euclidean distance*, denoted $\Delta_j^{(2)}$. In order to preserve the metric units, we use the square root this quantity, denoted Δ_j'. In short, we modified the definitions of Step 5 by computing the following values:

$$\begin{aligned}
\Delta_j^{(2)} &= \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|^2, \qquad \text{for } 1 \le j \le k \\
\Delta_j' &= \sqrt{\Delta_j^{(2)}}, \qquad \text{for } 1 \le j \le k \\
\Delta' &= \frac{1}{n} \sum_{j=1}^{k} n_j \Delta_j'.
\end{aligned}$$

The decision as to whether the split a cluster in Step 8 is instead based on the relative sizes of Δ'_j and Δ', rather than Δ_j and Δ. Note that this can produce different results. Nonetheless, based on many executions on both synthetically generated data and real images, we have found that our algorithm's actual performance is quite similar to that of ISOCLUS, in terms of the number of clusters obtained and the positions of their centers. This will be demonstrated in the next section. Thus, we believe that this modification does not significantly alter the nature of the algorithm, and has the benefit of running significantly faster. The value Δ'_j can be computed as outlined in Lemma 3.1. In the next section we present these experimental results. In Section 6, we provide theoretical justification for our modification.

5 Experiments

In order to establish the efficiency of our new algorithm and to determine the degree of similarity in clustering performance with the existing ISOCLUS algorithm, we ran a number of experiments on both synthetic and real data sets from remotely sensed images. Our modified algorithm involves changing both the functionality and computational approaches. To make the comparisons clearer, we implemented an intermediate, or *hybrid*, algorithm, which is functionally equivalent to one but uses the same computational approaches as the other.

Standard version (Std): The straightforward implementation of ISOCLUS as described in [28], which uses average Euclidean distances in Steps 5 and 8.

Hybrid version (Hyb): A modification of the standard version using Δ'_j and Δ' rather than Δ_j and Δ in Steps 5 and 8, but without using the filtering algorithm.

Filtering version (Fil): The same modification, but using the filtering algorithm for greater efficiency.

The Hybrid and Filtering versions are functionally equivalent, but use different computational approaches. The Standard and Hybrid versions are roughly equivalent in terms of the computational methods, but are functionally distinct. Our goal is to show that the Standard and Hybrid versions nearly functionally equivalent, and that the Filtering version is more efficient. All experiments were run on a SUN Blade 100 running Solaris 2.8, using the g++ compiler (version 2.95.3).

5.1 Synthetic data

We ran three sets of experiments on synthetically generated data sets to analyze the performance of our algorithm. For the first set of experiments we generated $n = 10,000$ data points in 3, 5, and 7 dimensional spaces. In each case the points were sampled with equal probability from a variable number of Gaussian clusters ranging from 10 to 100. The cluster centers were sampled uniformly at random from a hypercube of side length 2. In order to generate a point for each cluster, a vector is generated whose coordinates are drawn from a Gaussian distribution with a given standard deviation $\sigma = (1/k)^{1/d}$. The value of σ was derived by the following reasoning. In order for results to be comparable across different dimensions and with different numbers of clusters, it is desirable that clusters have comparable degrees of overlap. In low dimensions, the a significant amount of the probability mass of a Gaussian cluster lies within a region of volume $(2\sigma)^d$. We wish to subdivide a

cube of unit volume uniformly into k clusters, which suggests that each cluster should own $1/k$-th of the total volume, that is, σ should be chosen so that $(2\sigma)^d = 2^d/k$, from which obtain the above value of σ.

In the second set of experiments we randomly generated 100, 500, 1000, 5000, and 10,000 data points in 3-dimensional space all distributed evenly among 20 Gaussian clusters with average standard deviation of points and standard deviation of standard deviation among clusters following the same rule as mentioned above. We repeated each experiment in 5 and 7 dimensional space as well.

The third set of experiments, we varied both number of randomly generated points and number of clusters. We generated 100, 500, 1000, 5000, and 10,000 points such that they were distributed evenly among 5, 10, 20, 40, and 80 Gaussian clusters accordingly in 3, 5, and 7 dimensional space.

We ran the ISOCLUS algorithms for a maximum of 20 iterations ($I_{\max} = 20$), maximum cluster standard deviation of twice the value of the distribution standard deviation ($\sigma_{\max} = 2\sigma$), minimum cluster separation of 0.001 ($L_{\min} = 0.001$), minimum cluster size of $n_{\min} = n/(5k_{\text{init}})$. For the first set of experiments where $n = 10,000$ was fixed, we set the initial number of clusters to 10, 20, 40, 80, and 100 respectively. In each case, results were averaged over five runs. Results of these 3 sets of experiments are shown in Tables 1, 2, and 3.

Table 1: Results of Synthetic Data Test Inputs: n=10,000.

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
3	10	10	10	0.385	0.385	5.00	5.14	1.420	3.62
	20	20	20	0.286	0.286	8.74	9.07	1.788	5.07
	40	40	40	0.120	0.120	16.39	17.20	2.022	8.50
	80	78	78	0.077	0.077	33.87	34.99	2.626	13.32
	100	100	100	0.061	0.061	43.34	44.95	2.956	15.21
5	10	9	9	2.108	2.108	6.86	6.77	3.092	2.189
	20	19	19	1.184	1.184	12.58	13.20	3.880	3.403
	40	36	36	0.819	0.819	21.79	22.83	5.372	4.249
	80	79	79	0.490	0.490	48.69	50.01	7.998	6.253
	100	93	93	0.478	0.478	57.67	59.30	9.172	6.465
7	10	10	10	4.062	4.062	9.18	9.38	5.29	1.771
	20	20	20	1.971	1.971	16.31	16.82	7.40	2.274
	40	32	32	2.303	2.303	26.11	26.59	11.50	2.312
	80	74	74	1.447	1.447	59.27	60.29	22.07	2.732
	100	93	93	1.242	1.242	75.27	76.55	26.02	2.942

For each run, we computed the running time in CPU seconds, the final number of centers, and the final average distortion. Since the hybrid and filtering versions implement the same functional specifications, the final numbers of centers and final distortions are almost identical. (Small differences were observed, due to floating point round-off errors.) So, we list them together in the table (under the heading "Hyb/Fil"). We also computed the *speed-up*, which is defined as the ratio between the CPU times of the hybrid and the filtering versions.

In support of our claim that using squared distances does not significantly change the algorithm's clustering performance, observe that both algorithms performed virtually identically with respect

Table 2: Results of Synthetic Data Test Inputs: $k_{\text{init}} = 20$.

Dim	n	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
3	100	20	20	0.164	0.164	0.100	0.088	0.048	1.833
	500	20	20	0.278	0.278	0.446	0.428	0.148	2.892
	1000	20	20	0.265	0.265	0.902	0.876	0.256	3.422
	5000	20	20	0.288	0.288	4.512	4.232	0.960	4.408
	10000	20	20	0.286	0.286	9.290	8.804	1.770	4.974
5	100	20	20	0.828	0.828	0.138	0.116	0.092	1.261
	500	17	17	1.095	1.095	0.560	0.556	0.286	1.944
	1000	20	20	1.074	1.074	1.368	1.300	0.600	2.167
	5000	19	19	1.188	1.188	6.304	6.130	2.234	2.744
	10000	19	19	1.184	1.184	12.958	12.812	3.868	3.312
7	100	20	20	1.349	1.349	0.168	0.158	0.112	1.411
	500	17	17	1.957	1.957	0.690	0.692	0.478	1.450
	1000	18	18	1.990	1.990	1.546	1.526	0.924	1.652
	5000	19	19	1.990	1.990	8.078	7.994	4.146	1.928
	10000	20	20	1.971	1.971	16.740	16.604	7.472	2.222

Table 3: Results of Synthetic Data Test Inputs: Both n and k_{init} vary.

Dim	n	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
			Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
3	100	5	5	5	0.480	0.480	0.028	0.03	0.020	1.500
	500	10	10	10	0.357	0.357	0.236	0.24	0.100	2.380
	1000	20	20	20	0.265	0.265	0.870	0.91	0.232	3.931
	5000	40	40	40	0.120	0.120	8.082	8.50	1.158	7.344
	10000	80	78	78	0.077	0.077	34.074	35.33	2.682	13.174
5	100	5	5	5	2.350	2.350	0.036	0.04	0.030	1.334
	500	10	8	8	2.095	2.095	0.280	0.29	0.200	1.450
	1000	20	20	20	1.074	1.074	1.280	1.27	0.608	2.086
	5000	40	39	39	0.797	0.797	11.904	12.12	3.208	3.778
	10000	80	79	79	0.490	0.490	48.470	50.49	7.994	6.316
7	100	5	4	4	4.417	4.417	0.042	0.05	0.03	1.471
	500	10	9	9	4.321	4.321	0.398	0.42	0.33	1.282
	1000	20	18	18	1.990	1.990	1.550	1.58	0.90	1.750
	5000	40	36	36	2.201	2.201	14.782	15.07	7.06	2.135
	10000	80	74	74	1.447	1.447	46.966	60.69	22.04	2.753

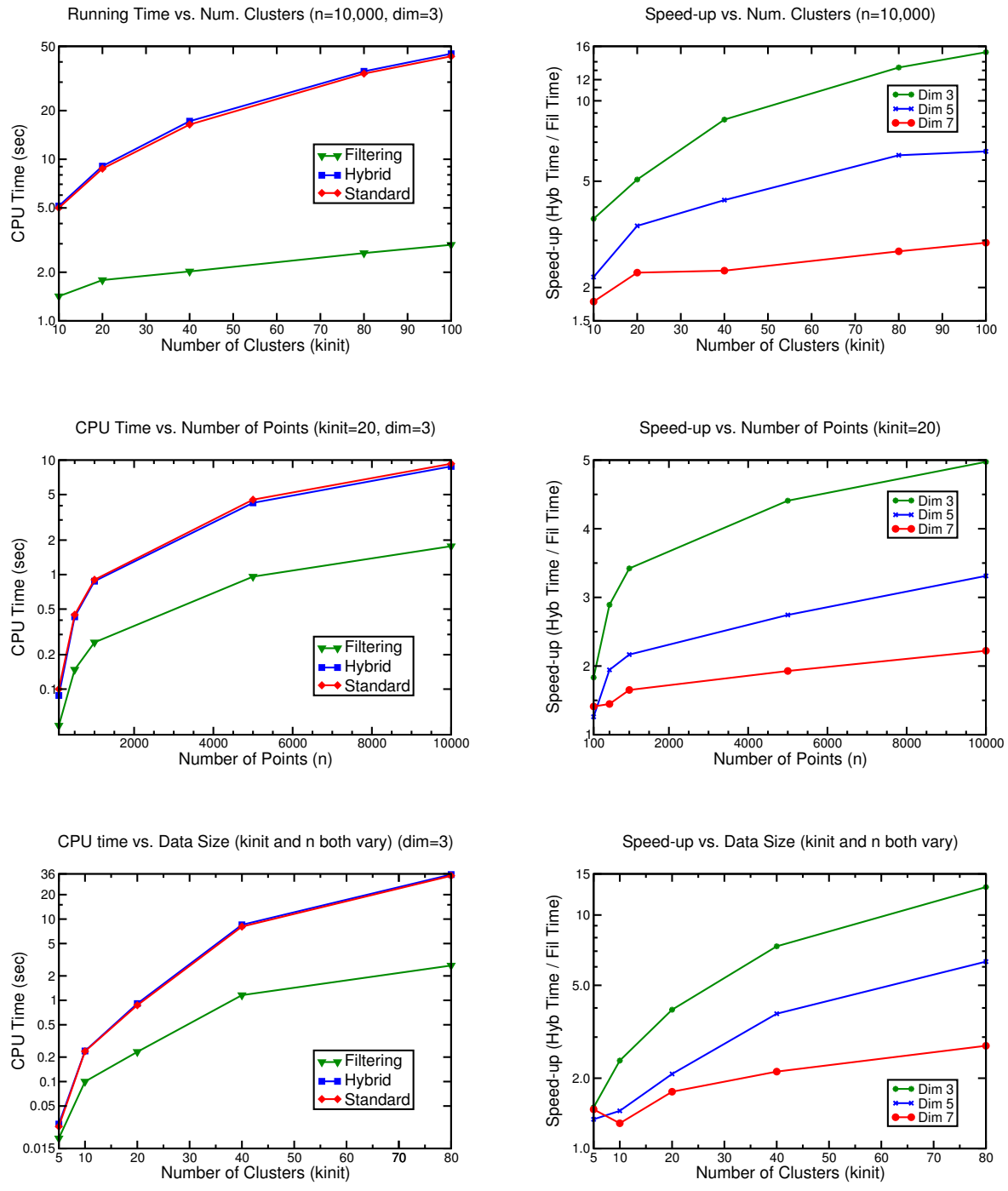

Fig. 5: CPU times and speed-ups for the various algorithms. (Note that the y-axis does not start at 0.)

to average distortions and the final number of centers. Also observe that the standard and hybrid versions run in roughly the same time, whereas the filtering version runs around 1.3 to 15.2 times faster than the other two. Fig. 5 shows our experimental results on synthetic data sets. We can see that for a fixed number of points, increasing the number of clusters increases both the CPU time and speed-up. The same result holds when we increase the number of points and fix the other parameters.

5.2 Image data

For image data we used two different data sets: Landsat and Modis data. For Landsat data we ran three tests on a 256×256 image of Ridgely, Maryland ($n = 65,536$). The first involved 3-dimensional data using bands 3-5, and the second was performed in 5-dimensional space using bands 3-7, and the third experiment was in 7-dimensional space and used all seven bands. We ran all three tests with three versions of ISOCLUS, each for 20 iterations and with 25 initial clusters, $\sigma_{\max} = 15$ and $L_{\min} = 10$ and $n_{\min} = 525$ (approximately $n_{\min} = n/(5k_{\text{init}})$). The results are presented in Table 4. As with the synthetic tests, all versions performed essentially equivalently with respect to the number of centers and final distortions. The filtering version was faster by a factor of roughly 7,5, and 3 accordingly. Fig. 6 shows the original data, and clustered images of standard and filtering ISOCLUS accordingly for the 3-dimensional case for which clustering results are identical.

For Modis data we ran three experiments on 128×128 ($n = 16384$)image taken from Agricultural area with the Konza Prairie in the state of Kansas, whose data were acquired in August 2001.

Like Landsat data, we tried experimenting in 3, 5, and 7-dimensional space. This time, we ran the experiments on 3, 5, and 7 principle components of the original raw data. Parameters were pretty much the same as for Landsat data except we started with 75 initial number of clusters and $n_{\min} = 45$. Final results for the 3-dimensional and 5-dimensional cases were identical with respect to both final number of clusters and final distortions. In 7-dimensional case, while all versions of the algorithm resulted in equal final number of clusters, their distortions were slightly different. In this case, only 3 percent of pixels were not appointed to the same clusters.

Table 4: Results of Landsat Data Test Inputs.

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
3	25	10	10	44.5	44.5	55.91	54.41	7.57	7.186
5	25	9	9	118.2	118.2	88.35	87.12	15.99	5.450
7	25	12	12	128.2	128.2	124.50	121.92	39.64	3.075

5.3 Experiments on Approximate Filtering

In order to better understand the effect of approximation, we experimented the approximate version of the filter-based algorithm. Recall that the algorithm differs from the exact algorithm in how candidate centers are pruned from each node in the process of determining which center is closest to the points of a node. The user supplies a value $\epsilon > 0$, and the algorithm may assign a point

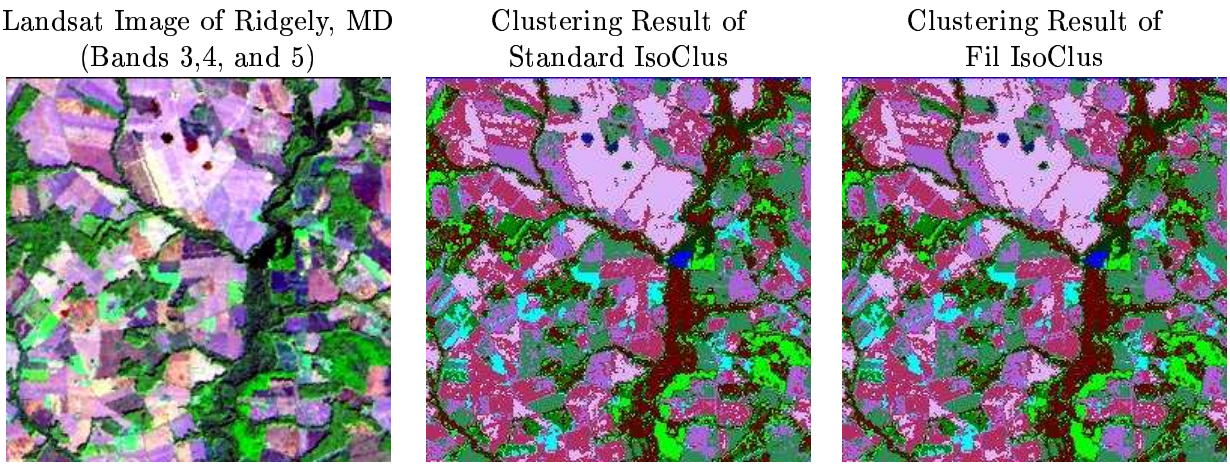

Fig. 6: Landsat image and classifications.

Table 5: Results of Modis Data Test Inputs.

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
3	75	77	77	137.9	137.9	63.12	60.94	3.47	17.58
5	75	97	97	403.6	403.6	119.80	116.62	9.00	12.96
7	75	103	103	645.5	644.5	197.12	195.18	20.36	9.59

to a center whose distance is at a distance of $(1+\epsilon)$ greater than the distance to its true nearest center. We performed experiments on both synthetic and satellite image data. In all cases, we ran experiments with approximation parameter $\epsilon \in \{0.1, 0.2, 0.5, 1.0, 1.5\}$, and compared the results against the exact ($\epsilon = 0$) case. Note that approximation was used in all but the last iteration of the algorithm, in which case exact pruning was performed. The reason is that, when the algorithm terminates, we want all the points to be assigned to their true closest center.

The use of ϵ values greater than 1 may seem to be unreasonably large for practical purposes, since this is allowing more than 100% relative error. But note that the ϵ value is an upper bound on the error committed per assignment. As we shall see below, even for fairly large values of ϵ, the observed distortions relative to the exact version of ISOCLUS were almost always less than 5%.

As mentioned at the end of Section 3.5, ISOCLUS is a heuristic and not an optimization algorithm. Thus, minor changes to the algorithm can result in convergence to local minima with a significantly different average distortions. This can happen even when $\epsilon = 0$, because the algorithm is started with random initial center points. For this reason, all of our experiments involved the average a number of runs of the algorithm.

For synthetic data, we generated 5 random sets of $n = 10,000$ points in 3, 5, and 7 dimensional space. Points were sampled with equal probability from 100 Gaussian clusters with uniformly distributed centers. This distribution and program parameter settings are the same as was described in the synthetic experiments of Section 5.1. We measured the CPU time, the final distortion, and the final number of clusters in each experiment. Finally, we evaluated the algorithm's performance with respect to results obtained by running the standard version of the ISOCLUS algorithm similarly

on the same data inputs. We calculated the speed-ups obtained as well as the relative errors with respect to standard version for overall distortions. These results are presented in Table 6.

For the satellite image data, we used the same Landsat and MODIS data sets with same parameter settings as described in Section 5.2. We used the same experimental setup as described above, but for each case, we ran the experiments 10 times, where each times a different set of initial centers were randomly selected at the beginning of the algorithm. The results are shown in Tables 7 and 8 for Landsat and MODIS data respectively.

The results show that approximation results in significant speed-ups for virtually all cases. In spite of the relatively large values of ϵ, the average relative error ("Dist Err %") in the final distortion relative to the exact case ($\epsilon = 0$) never exceeded 8% and was usually less than 3%. In addition, for the tougher case for the exact algorithm, namely for the synthetic data sets in dimension 7, the approximate algorithm showed that it is possible to achieve very speed-ups up to one order of magnitude with low average distortion errors. Note that increasing ϵ did not always lead to a decrease in execution time. This is because of ISOCLUS's sensitivity to its starting configuration, which further affects the number of iterations and number and structure of the clusters.

Table 6: Results of Synthetic Data Test Inputs: n=10,000, k_{init}= 100

Dim	ϵ	Final Ctrs.		Avg. Dist.×100		CPU Seconds		Speed-up	Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	98.2	98.2	6.09	6.09	43.12	2.72	15.85	0.00
	0.1		98.2		6.09		4.36	9.89	0.00
	0.2		98.4		6.09		3.90	11.06	0.00
	0.5		98.6		6.11		2.88	14.97	0.33
	1.0		98.0		6.35		1.99	21.67	4.27
	1.5		97.4		6.57		1.61	26.78	7.88
5	0.0	94.6	94.6	47.20	47.20	58.44	8.84	6.61	0.00
	0.1		94.6		47.20		17.40	3.36	0.00
	0.2		94.8		47.11		14.39	4.06	-0.19
	0.5		95.6		47.10		9.40	6.22	-0.21
	1.0		96.8		47.58		5.89	9.92	0.81
	1.5		96.6		48.59		4.20	13.91	2.94
7	0.0	93.2	93.2	124.76	124.76	73.70	24.64	2.99	0.00
	0.1		93.2		124.76		48.63	1.52	0.00
	0.2		93.2		124.76		39.79	1.85	0.00
	0.5		93.6		124.62		23.45	3.14	-0.11
	1.0		93.0		126.36		12.19	6.05	1.28
	1.5		93.4		129.04		7.48	9.85	3.43

6 Average Distance and Average Distortion

As we have mentioned, our use of the square root of the average distortion as a measure of cluster dispersion is different from the ISOCLUS algorithm's use of average distance. Our experiments suggest that this modification does not make a significant difference in the quality of the resulting clustering. The place that the ISOCLUS algorithm uses the value of Δ_j is in determining in Step 8 whether to split a cluster. In particular, the jth cluster is split if $\Delta_j > \Delta$. Thus, it would be of

Table 7: Results of Landsat Data Test Inputs, $k_{\text{init}} = 25$

Dim	ϵ	Final Ctrs.		Avg. Distortion		CPU Seconds		Speed-up	Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	7.9	7.9	56.58	56.85	47.31	6.61	7.16	0.47
	0.1		7.9		58.03		6.54	7.23	2.56
	0.2		7.8		58.05		5.93	7.98	2.60
	0.5		7.9		57.28		5.35	8.84	1.24
	1.0		8.3		54.87		5.12	9.24	-3.02
	1.5		8.5		55.14		5.06	9.35	-2.55
5	0.0	9.7	9.7	114.32	114.43	88.82	14.17	6.27	0.10
	0.1		9.4		115.26		16.45	5.40	0.82
	0.2		9.3		116.82		13.96	6.36	2.19
	0.5		9.5		109.81		9.45	9.40	-3.95
	1.0		9.2		114.52		7.91	11.23	0.17
	1.5		8.5		116.87		7.79	11.40	2.23
7	0.0	10.8	10.9	139.85	137.75	115.51	20.06	5.76	-1.50
	0.1		11.0		135.88		28.36	4.07	-2.83
	0.2		10.8		137.38		24.36	4.74	-1.77
	0.5		11.2		135.47		17.00	6.79	-3.13
	1.0		10.3		137.34		10.86	10.64	-1.79
	1.5		9.0		145.46		9.64	11.98	4.01

Table 8: Results of Modis Data Test Inputs, $k_{\text{init}} = 75$

Dim	ϵ	Final Ctrs.		Avg. Distortion		CPU Seconds		Speed-up	Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	74.9	74.9	137.93	138.04	58.83	3.18	18.50	0.08
	0.1		74.6		138.59		4.44	13.25	0.48
	0.2		74.5		138.74		3.86	15.24	0.59
	0.5		74.3		141.23		2.74	21.47	2.39
	1.0		75.6		143.55		1.90	30.96	4.07
	1.5		76.8		145.28		1.71	34.40	5.33
5	0.0	93.5	93.5	412.43	412.80	112.62	8.76	12.86	0.09
	0.1		93.3		413.88		15.39	7.32	0.35
	0.2		93.2		414.00		12.69	8.87	0.38
	0.5		94.1		412.80		8.39	13.42	0.09
	1.0		98.7		410.71		5.76	19.55	-0.42
	1.5		104.9		401.72		4.59	24.54	-2.60
7	0.0	106.3	106.1	633.54	635.31	180.77	18.63	9.70	0.28
	0.1		105.8		635.66		32.81	5.51	0.33
	0.2		106.3		634.93		27.71	6.52	0.22
	0.5		105.5		636.08		17.52	10.32	0.40
	1.0		110.0		631.15		10.73	16.85	-0.38
	1.5		118.4		618.51		8.60	21.02	-2.37

interest to establish under what circumstances the following equivalence holds:

$$\Delta_j > \Delta \text{ (in standard ISOCLUS)} \quad \Longleftrightarrow \quad \Delta'_j > \Delta' \text{ (in filtering ISOCLUS).}$$

This raises an important question of whether our modification is in some sense the "right" way to modify the algorithm. To further motivate this question, it is worth observing that there are other reasonable ways of generalizing dispersion that can produce substantially different results. Consider, for example, if we did not take the square root of the distortion. Then large distortions would play a disproportionately greater influence on the average dispersion, and this would result on different clusters being split in Step 5 of the algorithm.

To see this, consider for example, the following simple 1-dimensional example. There are three well separated clusters, each consisting of an equal number of points, and drawn from three normal distributions in which the standard deviations are 1, 6 and 9, respectively. Suppose further that the algorithm has placed three centers, one at the mean of each cluster. If the number of points is large, then the three average Euclidean distances as computed by the standard version of ISOCLUS are very nearly $\langle 1, 6, 9\rangle$ and so the overall average is $\Delta = (1 + 6 + 9)/3 \approx 5.333$, implying that the both the clusters whose standard deviations are 6 and 9 would be eligible to be split in step 8 of the algorithm. However, if squared distances are used, then the average of the squared distances would be very close to $\langle 1, 36, 81\rangle$. The overall average would be $(1 + 36 + 81)/3 \approx 39.333$, implying that only the cluster of standard deviation 9 would be eligible to be split. (In both cases $v_{j,\max} = \Delta'_j$, and so the additional condition $v_{j,\max} > \sigma_{\max}$ of Step 5 may be assumed to be satisfied in both cases.)

An alternative approach involves taking the square root of the of average distortion for each cluster (as we do in the filtering algorithm) and then taking the overall average dispersion as the square root of the weighted average of the squared distortions over all the clusters. (This is in contrast to the filtering algorithm, which takes square roots before averaging.) However, this alternative suffers from the same problem as the above approach.

Although it does not seem to be possible to make any theoretical assertions about the similarity between the results of the standard ISOCLUS algorithm and our modified version, we will endeavor to show that, in the limit, the approach taken in the filtering algorithm does not suffer from the biases of the above alternatives. Our statistical analysis is based on a number of fairly strong assumptions. Nonetheless, these assumptions are satisfied in the above examples, where the alternative definitions are shown to fail.

We assume that the point set S is drawn from k distinct cluster distributions in $\mathbb{R}^d$. We assume that all the cluster distributions are statistically identical up to a translation and uniform scaling. In particular, let $f(x_1, \ldots, x_d)$ be a d-variate probability density function [8] of the *base cluster distribution*, and let $\mathbf{X}$ denote a random vector sampled from this distribution. Without loss of generality, we may assume that its expected value, $E[\mathbf{X}]$, is the origin. Let $Y = \|\mathbf{X}\|$ be a random variable whose value is the Euclidean length of a vector drawn from this distribution. For the purposes of our analysis, we do not need to make any more specific assumptions about the base distribution. For example the distribution could be a Gaussian distribution centered about the origin with an arbitrary covariance matrix.

For $1 \leq j \leq k$, we assume that the points of the jth cluster are sampled from a distribution that arises by uniformly scaling all the coordinates of $\mathbf{X}$ by some positive scale factor $a_i \in \mathbb{R}^+$ and translated by some vector $\mathbf{t}_j \in \mathbb{R}^d$. Thus, a point of the jth cluster is generated by a random vector

$\mathbf{X}_j = a_i\mathbf{X} + \mathbf{t}_j$. Since the origin is the mean of the base distribution, $\mathbf{t}_j$ is the mean of the jth cluster, which we will call the *distribution center*. Let $Y_j = \|\mathbf{X}_j - \mathbf{t}\|$ be the random variable that represents the Euclidean distance from a point of the jth cluster to $\mathbf{t}_j$. Because this is a uniform scaling of the base distribution by a_i and translation by $\mathbf{t}_j$, it is easily verified that $E[Y_j] = a_i E[Y]$ and $E[Y_j^2] = a_i^2 E[Y^2]$.

We make the following additional assumptions about the clusters and the current state of the algorithm's execution.

(1) The clusters are well separated, meaning that the probability that a point belonging to one cluster is closer to the center of another cluster than to its own center is negligible.
(2) The number of points n_j in each cluster is sufficiently large that the law of large numbers can be applied.
(3) The algorithm is near convergence, in the sense that the difference between the current location of cluster center $\mathbf{z}_j$ and the actual cluster center $\mathbf{t}_j$ is negligible.

Theorem 6.1 *Subject to Assumptions* (1)–(3) *above, standard* ISOCLUS *and the filtering variant behave identically.*

Proof: As mentioned earlier, the only differences in the two algorithms are the computations of individual and average cluster dispersion in Step 5 and their use in determining whether to split a cluster in Step 8. Consider a cluster center j, for $1 \le j \le k$. Recall that to establish the equivalence of the two algorithms it suffices to show that

$$\Delta_j > \Delta \text{ (in standard ISOCLUS)} \quad \Longleftrightarrow \quad \Delta_j' > \Delta' \text{ (in filtering ISOCLUS)}.$$

First let us consider the average Euclidean distance of the standard algorithm. Recall that n_j denotes the number of points in a cluster. From the definitions of the cluster distributions and Assumption (3) we have

$$\Delta_j \;=\; \frac{1}{n_j}\sum_{\mathbf{x}\in S_j} \|\mathbf{x} - \mathbf{z}_j\| \;\approx\; \frac{1}{n_j}\sum_{\mathbf{x}\in S_j} \|\mathbf{x} - \mathbf{t}_j\|,$$

where $\approx$ denotes approximate equality (subject to the degree to which Assumption (3) is satisfied).

The elements of S_j are the points that are closer to $\mathbf{z}_j$ than to any other cluster center. By Assumptions (1) and (3), it follows that the contribution of points of S_j that arise from other clusters is negligible. From Assumption (2) it follows from the law of large numbers that this quantity, which is just a sample mean of a large number of independent and identically distributed random variables, will be arbitrarily close to the expected value for the cluster distribution. Thus we have

$$\Delta_j \;\approx\; E[\,\|\mathbf{X_j} - \mathbf{t}_j\|\,] \;=\; E[Y_j] \;=\; a_j E[Y].$$

Next, consider the average squared distance of the filtering algorithm. From Assumption (3), the corresponding quantity in this case is

$$\Delta_j' \;=\; \sqrt{\Delta_j^{(2)}} \;=\; \left(\frac{1}{n_j}\sum_{\mathbf{x}\in S_j} \|\mathbf{x} - \mathbf{z}_j\|^2\right)^{1/2} \;\approx\; \left(\frac{1}{n_j}\sum_{\mathbf{x}\in S_j} \|\mathbf{x} - \mathbf{t}_j\|^2\right)^{1/2}.$$

As before, from our assumptions we may approximate this sample mean with the expected value for the cluster distribution, from which we obtain

$$\Delta'_j \;\approx\; \sqrt{E[\,\|\mathbf{X}_j - \mathbf{t}_j\|^2\,]} \;=\; \sqrt{E[Y_j^2]} \;=\; \sqrt{a_j^2 E[Y^2]} \;=\; a_j\sqrt{E[Y^2]}.$$

Now, let us consider the average dispersions computed by the two algorithms. Let $w_j = n_j/n$ denote the fraction of points of S that are in cluster S_j. By the definitions of Δ and Δ' we have

$$\Delta \;=\; \frac{1}{n}\sum_{i=1}^{k} n_i\Delta_i \;=\; \sum_{i=1}^{k} w_i\Delta_i \qquad \text{and} \qquad \Delta' \;=\; \frac{1}{n}\sum_{i=1}^{k} n_i\Delta'_i \;=\; \sum_{i=1}^{k} w_i\Delta'_i.$$

Finally, we combine all of this to obtain the desired conclusion. Observe that the implications are not absolute, but hold in the limit as Assumptions (1)–(3) are satisfied:

$$\begin{array}{lllll}
\Delta_j > \Delta & \iff & \Delta_j > \sum_{i=1}^{k} w_i\Delta_i & \iff & a_jE[Y] > \sum_{i=1}^{k} w_i a_i E[Y] \\
 & & & & \Updownarrow \\
 & & & & a_j > \sum_{i=1}^{k} w_i a_i \\
 & & & & \Updownarrow \\
\Delta'_j > \Delta' & \iff & \Delta'_j > \sum_{i=1}^{k} w_i\Delta'_i & \iff & a_j\sqrt{E[Y^2]} > \sum_{i=1}^{k} w_i a_i\sqrt{E[Y^2]}
\end{array}$$

This completes the proof. □

7 Conclusion

We have demonstrated the efficiency of a new implementation of the ISOCLUS algorithm, based on the use of the kd-tree data structure and the filtering algorithm. Our algorithm is a slight modification of the original ISOCLUS algorithm, because it uses squared, rather than unsquared, Euclidean distances as a measure of cluster dispersion in determining whether to split clusters. We have provided both experimental and theoretical justification that the use of squared distances yields essentially the same results. The experiments on synthetic clustered data showed speed-ups in running times ranging from 1.3 to 15, while the experiments on Landsat image data showed speed-ups of 3 to 7, and experiments on Modis image data showed speed-ups of 9 to 17. We also presented an approximate version of the algorithm which allows the user to further improve the running time, at the expense of lower fidelity in computing the nearest cluster center to each point. We showed that with relatively small distortion errors, significant additional speed-ups can be achieved by this approximate version.

One possible direction for future research involves sensitivity to the input parameters. Our results indicate an interesting tradeoff in the sensitivity of the three algorithms to the various input size parameters. The hybrid and standard versions' running times increase linearly with the number of points n, the number of centers k, and the dimension d. For the inputs tested, the filtering version's running time increases sub-linearly in n and k, but super-linearly in the dimension d. Thus, the filtering version is most appropriate when n and k are large and dimension is small.

Acknowledgments

We would like to thank Jeff Morisette of NASA/GSFC and the EOS Validation Core Sites project for providing us with the MODIS data. We would also like to thank In-Joon Chu for his contributions to the geometrical analysis used in approximate filtering.

References

[1] Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. The discrete 2-center problem. In *Proc. 13th Annual ACM Sympos. Comput. Geom.*, pages 147–155, 1997.

[2] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-median and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 106–113, Dallas, TX, May 1998.

[3] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. Local search heuristics for k-median and facility location problems. In *Proc. 33rd Annual ACM Sympos. Theory of Comput.*, pages 21–29, Crete, Greece, 2001.

[4] G. H. Ball and D. J. Hall. Some fundamental concepts and synthesis procedures for pattern recognition preprocessors. In *Intl. Conf. on Microwaves, Circuit Theory, and Inform. Theory*, Tokyo, Japan, September 1964.

[5] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.

[6] L. Bottou and Y. Bengio. Convergence properties of the k-means algorithms. In G. Tesauro and D. Touretzky, editors, *Advances in Neural Information Processing Systems 7*, pages 585–592. MIT Press, 1995.

[7] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annual ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[8] W. Feller. *An introduction to probability theory and its applications.* John Wiley & Sons, New York, NY, 3rd edition, 1968.

[9] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768, 1965.

[10] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[12] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic, Boston, MA, 1992.

[13] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[14] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annual ACM Sympos. Theory Comput.*, pages 291–300, Chicago, IL, 2004.

[15] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, Boston, MA, 1997.

[16] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[17] J. R. Jensen. *Introductory Digital Image Processing: A Remote Sensing Perspective.* Prentice Hall, Upper Saddle River, NJ, second edition, 1996.

[18] T. Kanungo, D. M. Mount, N. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom. Theory Appl.*, 28:89–112, 2004.

[19] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:881–892, 2002.

[20] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, New York, NY, 1990.

[21] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York, NY, 3rd edition, 1989.

[22] S. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. In J. Nesetril, editor, *Proceedings of the Seventh Annual European Symposium on Algorithms*, volume 1643 of *Lecture Notes Comput. Sci.*, pages 362–371. Springer-Verlag, July 1999.

[23] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inform. Theory*, 28:129–137, 1982.

[24] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Stat. Prob.*, volume 1, pages 281–296, Berkeley, CA, 1967.

[25] O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1:183–201, 1997.

[26] J. Matoušek. On approximate geometric k-clustering. *Discrete and Computational Geometry*, 24:61–84, 2000.

[27] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 144–155, Santiago, Chile, September 1994.

[28] PCI Geomatics Corp. ISOCLUS–Isodata clustering program. `http://www.pcigeomatics.com/cgi-bin/pcihlp/ISOCLUS`.

[29] D. Pollard. A centeral limit theorem for k-means clustering. *Annals of Probability*, 10:919–926, 1982.

[30] J.A. Richards and X. Jia. *Remote Sensing Digital Image Analysis*. Springer, Berlin, 1999.

[31] S. Z. Selim and M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans. Patt. Anal. Mach. Intell.*, 6:81–87, 1984.

[32] Micha Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete Comput. Geom.*, 18:125–134, 1997.

[33] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, London, 1974.